# COMPONENT ARCHITECTURE

## CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. Patent Application No. 10/443,402 filed May 22, 2003, entitled "COMPONENT ARCHITECTURE" which application is a continuation of U.S. Patent Application No. 10/238,982 filed September 10, 2002, entitled "COMPONENT ARCHITECTURE," which application is a continuation of U.S. Patent Application No. 10/118,587 filed April 8, 2002, entitled "COMPONENT ARCHITECTURE," which application claims the benefit of U.S. Provisional Application No. 60/282,100, filed April 6, 2001, entitled "COMPONENT ARCHITECTURE," which applications are incorporated herein by reference in their entirety.

## FIELD OF THE INVENTION

The present invention is directed to a component architecture.

## SUMMARY

An audio server system for streaming audio content has an audio switch that receives commands from an audio control component and sends commands to an audio stream source, that receives events from an audio stream source and sends events to an audio browse list component, and that controls the switching of audio from the audio stream source to one or more receivers; and an audio control component that receives commands from an audio control synchronous component and sends commands to the audio switch.

## BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a high level overview of all of the components of a Vulcan architecture and the links between those components;

Figure 2 illustrates an overview of a component architecture; and

Figures 3-5 show screenshots of an exemplary Audio Browser.

## DETAILED DESCRIPTION

***Table of Contents***

~~DETAILED DESCRIPTION~~

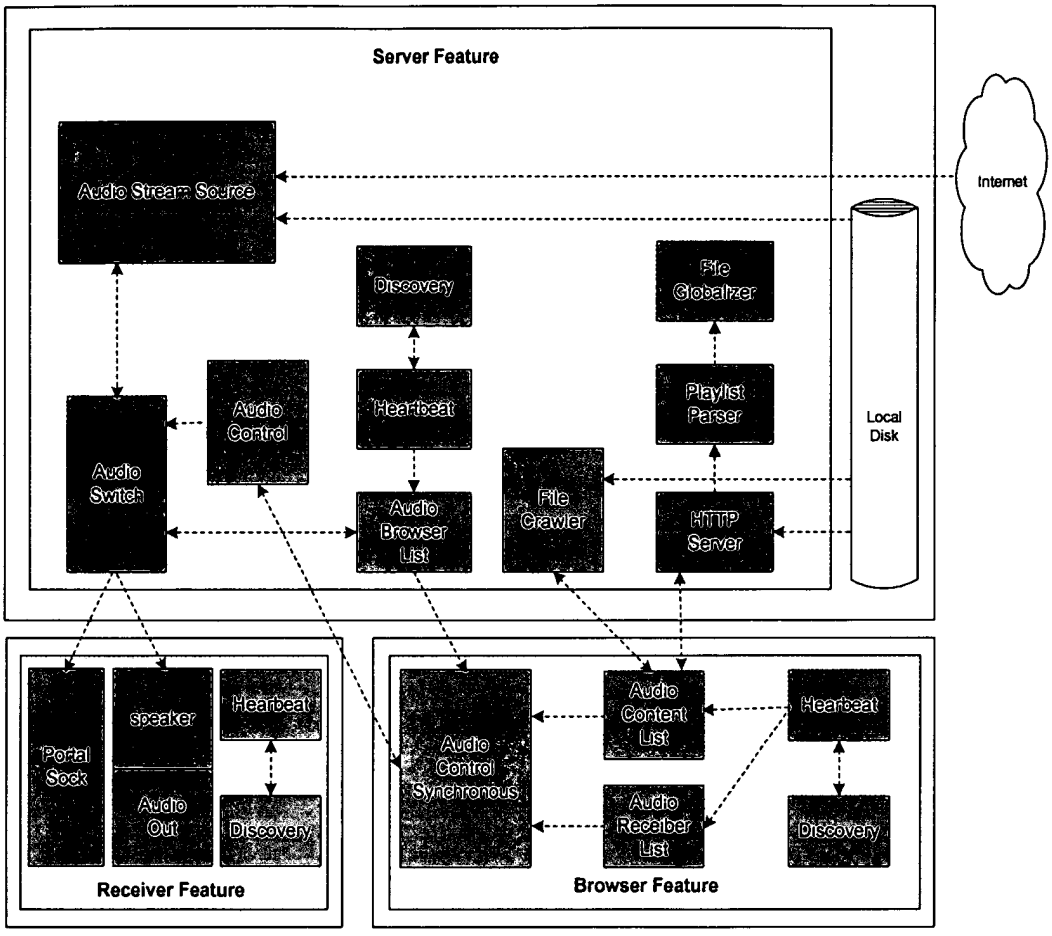
# 2 Design Overview

The Vulcan product which is designed to stream audio from a given audio content server PC to multiple connected audio receivers, is made up of a collection of components, i.e. Beads, which will communicate through command requests and event responses. This design will outline the structure of all commands and events in the system. In addition, the structure of any required protocol initialization files will also be designed.

# 3 Process Summary

The ~~following~~ diagram of Figure 1 details the high level overview of all of the components of the Vulcan architecture and the links between those components. ||

**Component Architecture Overview**




The interface between each component is detailed in the Component Interfaces Design document which is available at: Component Interfaces.xls.

The basic responsibility of each component in the diagram above is as follows:

AudioStreamSource:

- Interface bead which receives commands from AudioSwitch and talks to audio source interface libraries such as Real Audio and Windows Media.
- Sends asynchronous events to AudioSwitch.

AudioSwitch:

- Receives commands from AudioControl and sends commands to AudioStreamSource.
- Receives events from AudioStreamSource and sends events to AudioBrowserList.
- Controls the switching of audio from AudioStreamSource to one or more Receivers.
- Requests playlist content (see: Playlist Parsers), upon receipt of a CreateActivity command that contains a Playlist or a SetPlaylist command.
- Receive commands on it's config edge to list all current activities that it owns.

AudioControl:

- Receives commands from AudioControlSynchronous and sends commands to AudioSwitch.
- Requests playlist content (see: Playlist Parsers), upon receipt of the ExpandPlaylist command from an Audio Browser.

AudioControlSynchronous:

- Receives events from AudioBrowserList, AudioContentList and AudioReceiverList.
- Sends commands to AudioControl on the server local to the requested content or on the server responsible for serving up internet content, when applicable.
- Generates unique ActvityIDs across all Audio Browsers, using IP address and sequential ID.

AudioBrowserList:

- Receives messages from Heartbeat about the addition/removal of Audio Browsers and requests the initial list of active activities from AudioSwitch when a new Audio Browser is detected.

- Receives events from AudioSwitch and sends events to all active Audio Browsers.

AudioContentList:

- Receives messages from Heartbeat about the addition/removal of Audio Servers.
- Users IP address of new server to get virtual roots from HTTPServer, and uses virtual roots to get audio content from FileCrawler.
- Forwards new (and removed) content events to AudioControlSynchronous.

AudioReceiverList:

- Receives messages from Heartbeat about the addition/removal of Audio Receivers and forwards those messages in the form of events to AudioControlSynchronous.

HTTPServer:

- Receive commands on it's config edge to add/remove/modify virtual roots.
- Receive a command on it's config edge to return contents of virtual roots table.

FileCrawler:

- Receive a command on it's config edge to return a list of files for a given HTTP virtual root (and it's subdirectories) for a given content type (i.e. extension).
- Transform all local files names into encoded URLs.

Playlist Parser:

- Receive a format specific playlist and convert it into a generic filelist.
- There will actually be one PlaylistParser for each support playlist type, (i.e. M3UParse, PLSParse, ASXParse, etc.).

FileGlobalizer:

- Transforms local file names into URLs. It will be used in conjunction with the Playlist Parsers to return file lists with URLs instead of just local file names.

Heartbeat:

- On startup, post a message to discovery. announcing the presence, or absence, of each Strings feature, i.e., Local-Content Server, Internet Content Server, Receiver or Browser.
- Receive messages from discovery channels for each feature and forward events for each existing feature.

# 4 Design Concepts

The architecture is made up features which can be distributed in any way across hosts in the network. The only requirement is that at most one instance of a given feature can be running on a host at a time. The features that make up the architecture are the Server, Receiver and Browser. Server features can be further broken down to support local content, internet content or both. Heartbeat is responsible for discovering all features and forwarding corresponding events appropriately. AudioBrowserList will receive events for Browser features, AudioContentList will receive events for Server features and AudioReceiverList will receive events for Receiver features.

All features can communicate asynchronously over a well known port using sockets, or synchronously via an HTTP request.

AudioControlSvnchronous will receive all accessible content and receivers (from AudioContentList and AudioReceiverList, respectively) and all current activities from

AudioBrowserList on startup. Subsequent events to AudioControlSynchronous will only contain deltas to the original data dump and will be initiated as follows:

- An event is sent from AudioBrowserList indicating a modification of an existing activity.
- An event is sent from AudioContentList indicating that a new PC came on-line with new playlist/audiofile info or an existing PC went off-line, thus removing existing playlist/audiofile info.
- An event is sent from AudioReceiverList indicating that a new audio receiver came on-line or an existing audio receiver went off-line.

When AudioContentList receives the event from Heartbeat announcing a new Server feature, it will request the virtual roots on that server from HTTPServer. Then, it will request the audio content for each virtual root, for each supported content type, from the FileCrawler on the newly discovered server. FileCrawler will be responsible for parsing each filename and replacing all local file prefixes with URLs. For example, if it finds a playlist with the URL: C:\My Documents\My Music\workout.m3u, it will need to change that to http:\\a.b.c.d\c\my documents\my music\workout.m3u, assuming the virtual root is set up such that c:\ maps to http:\\a.b.c.d\c. FileCrawler will get the virtual roots from the HTTPServer config edge.

AudioContentList will also be responsible for sending an event to AudioControlSynchronous which details which servers are serving local content and which are serving internet content. This will allow AudioControlSynchronous to request audio content from the correct server.

When AudioSwitch receives a request from AudioControl to open a playlist, it will request the contents of the playlist, which will invoke a content specific playlist bead (i.e. M3UParse, PLSParse, ASXParse, etc.), to parse the results returned from HTTPClient and pass on the

content independent results. The results from the playlist parser will be passed through FileGlobalizer to replace all local file names with encoded URLs.

# 5 XML Format Specification

The current assumption is that the structure of all commands and event interfaces will use an XML message format. However, all implementation should be done in a modular way such that this decision can be changed at a later date without major effect on the code.

## *5.1 Element List*

This section documents all XML elements supported in the system. Each element is shown with it's associated attributes and all optional attributes will be in *italics*. If all attributes of a particular element are optional, it must contain at least one attribute to be valid. Child elements of a given element are not shown.

<CONTENT_EVENT version=" 1.0" / >

<SERVER_EVENT version=" 1.0" / >

<RECEIVER_EVENT version=" 1.0" / >

<ACTIVITY_EVENT version=" 1.0" / >

<PLAYBACK_EVENT version=" 1.0" / >

<HEARTBEAT_EVENT version=" 1.0" / >

<ACTIVITY_COMMAND version=" 1.0" / >

<PLAYBACK_COMMAND version=" 1.0" / >

<RESCAN_RESPONSE version=" 1.0" / >

<HEARTBEAT_MESSAGE version=" 1.0" / >

<FILE_EXTENSIONS version=" 1.0" / >

<LIST_EXTENSIONS version=" 1.0" / >

<HOSTS version=" 1.0" / >

<VIRTUAL_ROOTS version=" 1.0" / >

<MULTICAST_GROUPS version=" 1.0" / >

<ADD_TRACKS/>

<REMOVE_TRACKS/>

<ADD_PLAYLISTS/>

<REMOVE_PLAYLISTS/>

<ADD_INTERNET_SERVERS/>

<REMOVE_INTERNET_SERVERS/>

<ADD_LOCAL_SERVERS/>

<REMOVE_LOCAL_SERVERS/>

<ADD_RECEIVERS/>

<REMOVE_RECEIVERS/>

<ADD_TARGETS/>

<REMOVE_TARGETS/>

<ADD_HOST/>

<REMOVE_HOST/>

<CREATE_ACTIVITY/>

<SET_PLAYLIST/>

<SET_MODE/>

<RESET_CONTENT/>

<VIRTUAL_ROOT name=" virtualRootName"
localRoot="localRootName"/>

<HOST ipAddress=" ipAddress"
*name*=" hostName"
*dnsName*=" dnsName" />

<FILE *url*="linkName" *fullName*="localName"/>

<IP_ADDRESS value="224 . 0. 0.xx" />

<MODE value="Liner | Random" />

<STATUS value="Success | Failure" />

<ERROR/>

<STRINGS_CODE value="errorValue" />

<ACTIVITY id=" id" name=" name" />

<STATE value =
"Idle | Connecting | Buffering | Playing | Paused |
Stopped | Closed | Error" />

<FILELIST />

<PLAYLIST />

<TRACK />

<EXTENSION /> value" extension" />

<PROGRESS value=" 100" units="percentage" />

<LENGTH value=" 0" units="milliseconds" />

<POSITION value=" 0" units="milliseconds" />

<TITLE name="Title" />

<ARTIST name="Artist" />

<ALBUM name="Album" />

<GENRE name=" Genre" />

<OPEN/>

<PLAY/>

STOP/>

NEXT/>

PREVIOUS/>

SEEK/> offset==" value" units="milliseconds" />

PAUSE/>

RESUME/>

CLOSE/>

## *5.2 Element Structure*

This section documents the structure of the supported XML elements, by outlining the parent-child relationships between all of the elements. Attributes of each element are not included in this section.

| Element | Valid Parent Elements | Valid Child Elements |
|---|---|---|
| ACTIVITY | ACTIVITY_COMMAND, ACTIVITY_EVENT | ADD_TARGETS, ADD_TRACKS, ALBUM, ARTIST, CLOSE, CREATE_ACTIVITY, ERROR, GENRE, LENGTH, NEXT, PAUSE, PLAY, PLAYLIST, POSITION, PREVIOUS, PROGRESS, REMOVE_TARGETS, REMOVE_TRACKS, RESET_CONTENT, RESUME, SET_MODE, SET_PLAYLIST, SEEK, STATE, STOP, TITLE, TRACK |
| ACTIVITY_COMMAND | NONE | ACTIVITY |
| ACTIVITY_EVENT | NONE | ACTIVITY, ERROR |
| ADD_HOST | HEARTBEAT__EVENT | HOST |
| ADD_INTERNET_SERVERS | SERVER_EVENT | HOST |
| ADD_LOCAL_SERVERS | SERVER_EVENT | HOST |
| ADD_PLAYLISTS | CONTENT_EVENT | FILE |
| ADD_RECEIVERS | RECEIVER_EVENT | HOST |
| ADD_TARGETS | ACTIVITY, CREATE_ACTIVITY | HOST |
| ADD_TRACKS | ACTIVITY, CONTENT_EVENT, CREATE_ACTIVITY | FILE |
| ALBUM | ACTIVITY, PLAYBACK_EVENT | NONE |
| ARTIST | ACTIVITY, PLAYBACK_EVENT | NONE |
| CLOSE | ACTIVITY | NONE |
| CONTENT_EVENT | NONE | ADD_PLAYLISTS, ADD_TRACKS,_ERROR, REMOVE_PLAYLISTS, REMOVE_TRACKS |
| CREATE_ACTIVITY | ACTIVITY | ADD_TARGETS,_ADD_TRACKS, SET_MODE,_SET_PLAYLIST |

| | | |
|---|---|---|
| ERROR | ACTIVITY, ACTIVITY_EVENT, CONTENT_EVENT, PLAYBACK_EVENT, RECEIVER_EVENT | STRINGS_CODE |
| EXTENSION | FILE_EXTESIONS, LIST_EXTENSIONS | NONE |
| FEATURE_AVAILABLE | HEARTBEAT_MESSAGE | NONE |
| FEATURE_NOT_AVAILABLE | HEARTBEAT_MESSAGE | NONE |
| FILE | ADD_PLAYLISTS, ADD_TRACKS, FILELIST, OPEN, PLAYLIST, REMOVE_PLAYLISTS, REMOVE_TRACKS, SET_PLAYLIST, TRACK | NONE |
| FILE_EXTENSIONS | NONE | EXTENSION |
| FILELIST | NONE | FILE |
| GENRE | ACTIVITY, PLAYBACK_EVENT | NONE |
| HEARTBEAT_EVENT | NONE | ADD_HOST, REMOVE_HOST |
| HEARTBEAT_MESSAGE | NONE | FEATURE_AVAILABLE, FEATURE_NOT_AVAILABLE |
| HOSTS | NONE | HOST |
| HOST | ADD_HOST, ADD_RECEIVERS, ADD_TARGETS, HEARTBEAT, HOSTS, REMOVE_HOST, REMOVE_RECEIVER, REMOVE_TARGETS | NONE |
| IP_ADDRESS | MULTI_CAST_GROUP S | NONE |
| LENGTH | ACTIVITY, PLAYBACK_EVENT | NONE |
| LIST_EXTENSIONS | NONE | EXTENSION |
| MODE | SET_MODE | NONE |
| MULTICAST_GROUPS | NONE | IP_ADDRESS |
| NEXT | ACTIVITY | NONE |
| OPEN | PLAYBACK_COMMAND | FILE |
| PAUSE | ACTIVITY, PLAYBACK_COMMAND | NONE |
| PLAY | ACTIVITY, PLAYBACK_COMMAND | NONE |
| PLAYBACK_COMMAND | NONE | OPEN, PAUSE, PLAY, RESUME, SEEK, STOP |
| PLAYBACK_EVENT | NONE | ALBUM, ARTIST, ERROR, GENRE, LENGTH, POSITION, PROGRESS, STATE, TITLE |
| PLAYLIST | ACTIVITY | FILE |

| | | |
|---|---|---|
| POSITION | ACTIVITY, PLAYBACK_EVENT | NONE |
| PREVIOUS | ACTIVITY | NONE |
| PROGRESS | ACTIVITY, PLAYBACK_COMMAND | NONE |
| RECEIVER_EVENT | NONE | ADD_RECEIVERS, ERROR, REMOVE_RECEIVERS |
| REMOVE_HOST | HEARTBEAT_EVENT | HOST |
| REMOVE_INTERNET_SERVERS | SERVER_RESPONSE | HOST |
| REMOVE_LOCAL_SERVERS | SERVER_RESPONSE | HOST |
| REMOVE_PLAYLISTS | CONTENT_EVENT | FILE |
| REMOVE_RECEIVERS | RECEIVER_EVENT | HOST |
| REMOVE_TARGETS | ACTIVITY | HOST |
| REMOVE_TRACKS | ACTIVITY, CONTENT_EVENT | FILE |
| RESCAN_RESPONSE | NONE | STATUS |
| RESET_CONTENT | ACTIVITY | NONE |
| RESUME | ACTIVITY, PLAYBACK_COMMAND | NONE |
| SEEK | ACTIVITY, PLAYBACK_COMMAND | NONE |
| SERVER_RESPONSE | NONE | ADD_INTERNET_SERVERS, ADD_LOCAL_SERVERS, REMOVE_INTERNET_SERVERS, REMOVE_LOCAL_SERVERS |
| SET_MODE | ACTIVITY, CREATE_ACTIVITY | MODE |
| SET_PLAYLIST | ACTIVITY, CREATE_ACTIVITY | FILE |
| STATE | ACTIVITY, PLAYBACK_EVENT | NONE |
| STATUS | RESCAN_RESPONSE | NONE |
| STOP | ACTIVITY, PLAYBACK_COMMAND | NONE |
| STRINGS_CODE | ERROR | NONE |
| TITLE | ACTIVITY, PLAYBACK_EVENT | NONE |
| TRACK | ACTIVITY | FILE |
| VIRTUAL_ROOT | VIRTUAL_ROOTS | NONE |
| VIRTUAL_ROOTS | NONE | VIRTUAL_ROOT |

# 6. Protocol Interface Specification

This section describes all protocol interfaces which utilize XML and query string message format. This includes all command, event and config edges as well as interfaces to discovery and initialization file formats.

## *6.1 AudioControlSynchronous*

AudioControlSynchronous supports an Event Interface on which it receives events from AudioBrowserList, AudioContentList and AudioReceiverList to drive the GUI and a config interface on which it receives commands from a web browser, however, the config interface is not included in this document. It also outputs commands to the AudioControl Command Interface and Config Interface.

### 6.1.1 Event Interface

This interface supports four separate events: content, server, receiver and activity events. Event event is described in detail below.

The first type of event is a content event which contains changes to the available content based on the availability of server features on the network. AudioControlSynchronous can support any combination of elements under the CONTENT_EVENT root element. AudioControlSynchronous will also need to be robust enough to support duplicate add and remove events, by ignoring all but the first event. The structure of a content event is as follows:

```
<?xml version="1.0"  encoding="UTF-8"  standalone="yes" ?>

<CONTENT_EVENT  version=" 1.0">

    <ERROR>

        <STRINGS_CODE          value=" errorValue" />

    </ ERROR>

    <ADD_TRACKS>
```

<FILE *url*=" linkName" *fullName*=" localName" />

<FILE *url*=" linkName" *fullName*=" localName" />

</ ADD_TRACKS>

<ADD_PLAYLISTS>

<FILE *url*=" linkName" *fullName*=" localName" />

<FILE *url*=" linkName" *fullName*=" localName" />

</ ADD_PLAYLISTS>

<REMOVE_TRACKS>

<FILE *url*=" linkName" *fullName*=" localName" />

<FILE *url*=" linkName" *fullName*=" localName" />

</ REMOVE_TRACKS>

<REMOVE_PLAYLISTS>

<FILE *url*=" linkName" *fullName*=" localName" />

<FILE *url*=" linkName" *fullName*=" localName" />

</ REMOVE_PLAYLISTS>

</CONTENT_EVENT>

The second type of event is a server event which contains changes to the available servers based on the availability of server features on the network, and whether a server supports local content, internet content or both. AudioControlSynchronous can support any combination of elements under the SERVER_EVENT root element. AudioControlSynchronous will also

need to be robust enough to support duplicate add and remove events, by ignoring all but the first event. The structure of a server event is as follows:

```
<?xml version="1.0"  encoding="UTF-8"  standalone="yes" ?>

<SERVER_EVENT version=" 1.0" >

    <ERROR>

        <STRINGS_CODE  value=" errorValue" />

    </ ERROR>

    </ADD_INTERNET_SERVERS>

        <HOST       ipAddress=" ipAddress"

                    name=" hostName"

                    dnsName=" dnsName" />

    </ADD_INTERNET_SERVERS>

    <ADD_LOCAL_SERVERS>

        <HOST       ipAddress=" ipAddress"

                    name=" hostName"

                    dnsName=" dnsName" />

    </ ADD_LOCAL_SERVERS>

    <REMOVE_INTERNET_SERVERS>

        <HOST       ipAddress=" ipAddress"

                    name=" hostName"

                    dnsName=" dnsName" />
```

</ REMOVE_INTERNET_SERVERS>

<REMOVE_LOCAL_SERVERS>

<HOST ipAddress=" ipAddress"

*name*=" hostName"

*dnsName*=" dnsName" />

</ REMOVE_LOCAL_SERVERS>

</ SERVER_EVENT>

The third type of event is a receiver event which contains changes to the available receivers based on the availability of receiver features on the network.

AudioControlSynchronous can support any combination of elements under the RECEIVER_EVENT root element. AudioControlSynchronous will also need to be robust enough to support duplicate add and remove events, by ignoring all but the first event. The structure of a receiver event is as follows:

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<RECEIVER_EVENT version=" 1.0" >

<ERROR>

<STRINGS_CODE value=" errorValue" />

</ERROR>

```
</ADD_RECEIVERS>
        <HOST       ipAddress=" ipAddress"
                    name=" hostName"
                    dnsName=" dnsName" />
</ ADD_RECEIVERS>

<REMOVE_RECEIVERS>
        <HOST       ipAddress=" ipAddress"
                    name=" hostName"
                    dnsName=" dnsName" />
</ REMOVE_RECEIVERS>

</ RECEIVER_EVENT>
```

The fourth type of event is an activity event which contains changes to the active activities in the system. An activity event can contain multiple ACTIVITY elements and an ACTIVITY element can contain any combination of child elements. The structure of an activity event is as follows:

```
<ACTIVITY_EVENT version=" 1.0" >

        <ERROR>
            <STRINGS_CODE value=" errorValue" />
        </ERROR>

        <ACTIVITY id=" id" name="name" >
```

<ERROR>

<STRINGS_CODE value=" errorValue" />

</ERROR>

<ADD_TARGETS>

<HOST ipAddress=" ipAddress"

*name*=" hostName"

*dnsName*=" dnsName" />

<ADD_TARGETS>

<REMOVE_TARGETS

<HOST ipAddress=" ipAddress"

*name*=" hostName"

*dnsName*=" dnsName" />

<REMOVE_TARGETS

<PLAYLIST>

<FILE *url*=" linkName" *fullName*=" localName" />

</ PLAYLIST>

<TRACK>

<FILE *url*=" linkName" *fullName*=" localName" />

</ TRACK>

<STATE value =

"Idle | Connecting | Buffering | Playing | Paused |

```
                    Stopped | Closed | Error" />

        <PROGRESS   value=" 100"  units="percentage" />

        <LENGTH     value=" 0"    units="milliseconds" />

        <POSITION   value=" 0"    units="milliseconds" />

        <TITLE      name="Title" />

        <ARTIST     name="Artist" />

        <ALBUM      name="Album" />

        <GENRE      name=" Genre" />

        <CLOSE/>

        </ACTIVITY>

    </ACTIVITY_EVENT>
```

Note that AudioControlSynchronous will also receive content events, server events, receiver events and activity events on startup to set the initial state of the system.

## *6.2 AudioControl*

AudioControl supports a Command Interface on which it receives commands from AudioControlSynchronous and a Config Interface that supports synchronous requests from AudioControlSynchronous. It also outputs commands to the AudioSwitch Command Interface.

### 6.2.1 Command Interface

The AudioControl command interface supports activity commands. Activity commands act on a specific activity and can contain at most one ACTIVITY element, but an ACTIVITY element can contain any combination of child elements. All activity commands require an ActivityId. The ActivityId is generated by AudioControlSynchronous and will be a combination of the IP address where the browser feature is running and a unique sequentional identifier. i.e. 10.1.1.20.1, 10.1.1.20.2, etc. Once the unique ActivityId is generated, AudioControlSynchronous can send multiple commands per activity as required. The complete structure is as follows:

```
<?xml version="1.0"      encoding="UTF-8"    standalone="yes" ?>

    <ACTIVITY_COMMAND  version=" 1.0" >

    <ACTIVITY id="id" name="name">

            <CREATE_ACTIVITY>

                <ADD_TARGETS>

                    <HOST       ipAddress=" ipAddress"

                                name=" hostName"

                                dnsName=" dnsName" />

                    <HOST       ipAddress=" ipAddress"

                                name=" hostName"

                                dnsName=" dnsName" />

                <ADD_TARGETS>
```

<SET_PLAYLIST>

<FILE *url*=" linkName" *fullName*=" localName" />

<SET_PLAYLIST>

<ADD_TRACKS>

<FILE *url*=" linkName" *fullName*=" localName" />

<FILE *url*=" linkName" *fullName*=" localName" />

<FILE *url*=" linkName" *fullName*=" localName" />

</ADD_TRACKS>

<SET_MODE>

<MODE value=" Linear 1 Random" />

</ SET_MODE>

</ CREATE_ACTIVITY>

<PLAY/>

<ADD_TRACKS>

<FILE *url*=" linkName" *fullName*=" localName" />

<FILE url=" linkName" fullName=" localName" />

</ ADD_TRACKS>

<REMOVE_TRACKS>

<FILE *url*=" linkName" *fullName*=" localName" />

*<FILE url=" linkName" fullName=" localName" />*

</ REMOVE_TRACKS>

<SET_PLAYLIST>

<FILE *url*=" linkName" *fullName*=" localName" />

</ SET_PLAYLIST>

<SET_MODE value="Linear | Random" />

<RESET_CONTENT/>

<STOP/>

<NEXT/>

<PREVIOUS

<SEEK offset=" value" units=" milliseconds" />

<PAUSE/>

<RESUME/>

<ADD_TARGETS>

<HOST ipAddress=" ipAddress"

*name*=" hostName"

*dnsName*=" dnsName" />

</ADD_TARGETS>

```
<REMOVE_TARGETS>
    <HOST       ipAddress=" ipAddress"
                name=" hostName"
                dnsName=" dnsName" />
</REMOVE_TARGETS>
<CLOSE/>
</ACTIVITY>
</ACTIVITY_COMMAND>
```

The CREATE ACTIVITY command described above has very specific semantics as follows:

| # of Playlists | # of Tracks | Mode | Description |
|---|---|---|---|
| 0 | 0 | Ignored | INVALID |
| 0 | 1 | Ignored | Play single track only. |
| 1 | 1 | Linear* | Play list starting at selected track. |
| 1 | 0 | Linear* | Play list starting at first track, continue sequentially until end. |
| 1 | 0 | Random | Play list starting at random track, continue randomly until all songs played. |
| 0 | 2+ | Linear* | Play all tracks in order received. |
| 0 | 2+ | Random | Play all tracks in random order. |
| 1 | 2+ | Ignored | INVALID |

* Default

### 6.2.2 Config Interface

There is one command supported by the AudioControl config interface, called ExpandPlaylist and the syntax is as follows:

http://ipAddress/portal/protocols/AudioControl?Command=ExpandPlaylis

t&Url=playlistUrl

ExpandPlaylist returns the contents of the requested playlist with all local file names translated into encoded URLs. The structure of the return message is as follows:

```
<?xml  version="1.0"      encoding="UTF-8"  standalone="yes" ?>

<FILELIST  version=" 1.0" >
        <FILE  url=" linkName"      fullName=" localName" />
        <FILE  url=" linkName"      fullName=" localName" />
        <FILE  url=" linkName"      fullName=" localName" />
        <FILE  url=" linkName"      fullName=" localName" />
</FILELIST>
```

## 6.3 AudioSwitch

AudioSwitch supports a Command Interface on which it receives commands from AudioControl, an Event Interface on which it receives events from AudioStreamSource and a Config Interface on which it receives synchronous command requests from AudioBrowserList. It also outputs commands to the AudioStreamSource Command Interface and activity events to the AudioBrowserList Event Interface.

### 6.3.1 Command Interface

AudioSwitch supports the same command interface as AudioControl, since AudioControl is just responsible for sending commands to the correct session of AudioSwitch. For the complete interface specification, see AudioControl Command Interface.

### 6.3.2 Event Interface

The data sent to AudioSwitch from AudioStreamSource via the AudioSwitch event interface captures changes in the current status of the AudioStreamSource session. AudioStreamSource will be sending events and audio data to AudioSwitch via the event edge, so all payload data, include the XML event messages will have an RTP header on them to differentiate the audio from the events. AudioSwitch can support any combination of elements under the PLAYBACK_EVENT root element. The complete interface is as follows:

```
<?xml version="1.0" encoding="UTF-8"    standalone="yes" ?>

<PLAYBACK_EVENT       version=" 1.0" >

    <ERROR>

        <STRINGS_CODE      value=" errorValue" />

    </ERROR>

    <STATE       value = | Buffering | Playing | Paused | Stopped |

                 Closed | Error" />

    <PROGRESS          value=" 100"   units="percentage" />
```

<LENGTH value=" 0" units="milliseconds" />

<TITLE name="Title" />

<ARTIST name="Artist" />

<ALBUM name="Album" />

<GENRE name=" Genre" />

</ PLAYBACK _EVENT>

### 6.3.3 Config Interface

There is one command supported by the AudioSwitch config interface, called ListActivities, which returns all of the currently active activities owned by AudioSwitch, and the syntax is as follows:

http://ipAddress/portal/protocols/AudioSwitch?Command=ListActivities

The structure of the return message is the same as the activity event as defined by the AudioBrowserList Event Interface.

## *6.4 AudioStreamSource*

AudioStreamSource supports a Command Interface on which it receives commands from AudioSwitch. It also outputs events to the AudioSwitch Event Interface.

### *6.4.1 Command Interface*

AudioStreamSource receives commands from AudioSwitch which manage the AudioStreamSource session and the audio being streamed by that session. AudioStreamSource can support any combination of elements under the PLAYBACK_COMMAND root element. The complete structure is as follows.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_COMMAND version=" 1.0" >

    <OPEN>

        <FILE  url=" linkName"  fullName=" localName" />

    </OPEN>

    <PLAY/>

    <STOP/>

    <SEEK  offset=" value"  units=" milliseconds" />

    <PAUSE/>

    <RESUME/> </ PLAYBACK_COMMAND>
```

### 6.4.2 Config Interface

There is one command supported by the AudioStreamSource config interface, called ListMetadata, which returns all available information about a URL, and the syntax is as follows:

http://ipAddress/portal/protocols/AudioStreamSource?Command=ListMetadata&Url=url

The structure of the return message is the same as the PLAYBACK EVENT as defined by the AudioSwitch Event Interface.

## *6.5 FileCrawler*

FileCrawler supports a Config Interface on which it receives synchronous commands from AudioContentList.

### 6.5.1 Config Interface

There is one command supported by the FileCrawler config interface, called ListFiles, which returns all of the files under the virtual root directory that match the given extension.

http://ipAddress/portal/protocols/FileCrawler?Command=ListFiles&Root=virtualRoot&Extension=extension

The structure of the return message is as follows:

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>.

```
<FILELIST  version=" 1.0" >
    <FILE   url=" linkName"        fullName=" localName" />
    <FILE   url=" linkName"        fullName=" localName" />
    <FILE   url=" linkName"        fullName=" localName" />
</ FILELIST>
```

## *6.6 AudioBrowserList*

AudioBrowserList supports an Event Interface on which it receives events from AudioSwitch with current activity information and it outputs activity events as specified by the AudioControlSynchronousEvent Interface. It also supports a Heartbeat Interface on which it receives events from Heartbeat announcing the addition or removal of new browser features to and from the network.

### 6.6.1 Event Interface

AudioBrowserList supports the same event interface as AudioControlSynchronous since AudioBrowserList is just responsible for multiplexing the events it receives to all connected browser features. For the complete interface specification, see AudioControlSynchronous Event Interface.

### 6.6.2 Heartbeat Interface

This interface receives a heartbeat event from Heartbeat announcing the the addition or removal of a host which is currently running the browser feature. The structure of a heartbeat event is as follows:

```
<HEARTBEAT_EVENT version=" 1 . 0">

    <ADD_HOST>

        <HOST    ipAddress=" ipAddress"

                 name=" hostName"

                 dnsName=" dnsName" />

    </ADD_HOST>

    <REMOVE_HOST>

        <HOST    ipAddress=" ipAddress"

                 name=" hostName"

                 dnsName=" dnsName" />

    </ REMOVE_HOST>

</ HEARTBEAT_EVENT>
```

## 6.7 AudioContentList

AudioContentList supports a Config Interface on which it receives synchronous commands and it outputs content events and server events as specified by the AudioControlSynchronous Event Interface. It also supports a Heartbeat Interface on which it receives events from Heartbeat announcing the addition or removal of new server features to and from the network. Lastly, it requires an initialization file to define the supported file and playlist types.

### 6.7.1 Config Interface

At a high level the AudioContentList config interface needs to support the ability to list all of the content currently accessible across the network. In addition, it needs to provide an interface to allow the user to rescan content that may have changed. This can be done globally, for a particular host or for a particular virtual root on a host.

The first command supported by the config interface is called ListContent, which returns all of the content currently available across all discovered server features.

http://ipAddress/portal/protocols/AudioContentList?Command=ListContent

The structure of the return message is as follows:

```
<?xml version="1.0"  encoding="UTF-8"    standalone="yes" ?>

<CONTENT_EVENT version="1.0">

    <ADD_TRACKS>

        <FILE      url=" linkName"      fullName=" localName" />

        <FILE      url=" linkName"      fullName=" localName" />

    </ ADD_TRACKS>

    <ADD_PLAYLISTS>

        <FILE      url=" linkName"      fullName=" localName" />

        <FILE      url=" linkName"      fullName=" localName" />

    </ ADD_PLAYLISTS>
```

</ CONTENT_EVENT>

The commands supported to rescan content are as follows:

http://ipAddress/portal/protocols/AudioContentList?Command=ListHosts

http://ipAddress/portal/protocols/AudioContentList?Command=ListVirtu
alRoots&Host=ipAddress

http://ipAddress/portal/protocols/AudioContentList?Command=Rescan

http://ipAddress/portal/protocols/AudioContentList?Command=Rescan&Host=ipAddress

http://ipAddress/portal/protocols/AudioContentList?Command=Rescan&Host=ipAddress
&VirtualRoot=virtualRoot

The first command will return all hosts currently accessible in the system. The structure of the return message is as follows:

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<HOSTS version=" 1. 0" >

<HOST ipAddress=" ipAddress"
*name*=" hostName"
*dnsName*=" dnsName" />

<HOST ipAddress=" ipAddress"
*name*=" hostName"
*dnsName*=" dnsName" />

```
</ HOSTS>
```

The second command will return all virtual roots for a given host. The structure of the return message is as follows:

```
<?xml version="1.0"  encoding="UTF-8"    standalone="yes"  ?>

<VIRTUAL_ROOTS version=" 1.0">

    <VIRTUAL_ROOT        name=" virtualRootName"
                         localRoot=" localRootName" />
    <VIRTUAL_ROOT        name=" virtualRootName"
                         localRoot=" localRootName" />
    <VIRTUAL_ROOT        name=" virtualRootName"
                         localRoot=" localRootName" />

</VIRTUAL_ROOTS>
```

The remaining three commands will scan all hosts, a particular host or a particular virtual root, respectively and will result in a CONTENT EVENT being sent as defined by the AudioControlSynchronous Event Interface. The config edge will return a status message as follows:

```
<?xml version="1.0"  encoding="UTF-8"    standalone="yes" ?>

<RESCAN_RESPONSE      version=" 1.0" >

    <STATUS     value=" Success | Failure" />
```

```
</RESCAN_RESPONSE>
```

### 6.7.2 Heartbeat Interface

This interface receives a heartbeat event from Heartbeat announcing the addition or removal of a host which is currently running the server feature. The structure of a HEARTBEAT EVENT is the same as defined by the AudioBrowserList Heartbeat Interface.

### 6.7.3 Initialization File

The AudioContentList.protocol initialization file, will contain data in XML format which describes what the supported audio content types are and which types are lists and which are single files. The XML format and will look like the following:

```
<?xml version='1.0'    encoding='UTF-8'      standalone='yes' ?>

<PROTOCOL name='AudioContentList'>

    <INITFILE module='AudioContentList'>

        <FILE_EXTENSIONS version="1.0">

            <EXTENSION        value=" mp3" />

            <EXTENSION        value=" wav" />

            <EXTENSION        value=" wma" />

        </FILE_EXTENSIONS>

        <LIST_EXTENSIONS  version="1.0">
```

```
<EXTENSION      value=" m3u" />
<EXTENSION      value=" pls" />
<EXTENSION      value=" asx" />
</ LIST_EXTENSIONS>
</ INITFILE>
</ PROTOCOL>
```

Note that this is just an example and does not represent a complete list.

## *6.8 AudioReceiverList*

AudioReceiverList supports a Config Interface on which it receives synchronous commands and it outputs receiver events as specified by the AudioControlSynchronous Event Interface. It also supports a Heartbeat Interface on which it receives events from Heartbeat announcing the addition or removal of new receiver features to and from the network.

### 6.8.1 Config Interface

There is one command supported by the AudioReceiverList config interface, called ListReceivers, which returns all of the discovered receivers.

http://ipAddress/portal/protocols/AudioReceiverList?Command=ListReceivers

The structure of the return message is as follows:

```
<?xml version="1.0"  encoding="UTF-8"    standalone="yes" ?>

<RECEIVER_EVENT version="1.0">

    <ADD_RECEIVERS>

        <HOST    ipAddress=" ipAddress"

                 name=" hostName"

                 dnsName=" dnsName" />

        <HOST    ipAddress=" ipAddress"

                 name=" hostName"

                 dnsName=" dnsName" />

        <HOST    ipAddress=" ipAddress"

                 name=" hostName"

                 dnsName=" dnsName" />

    </ADD_RECEIVERS>

</RECEIVER_EVENT>
```

### 6.8.2 Heartbeat Interface

This interface receives a heartbeat event from Heartbeat announcing the addition or removal of a host which is currently running the receiver feature. The structure of a HEARTBEAT EVENT is the same as defined by the AudioBrowserList Heartbeat Interface.

## *6.9 HTTPServer*

HTTPServer supports a Config Interface on which it receives synchronous commands from AudioContentList. It requires an initialization file to define the supported virtual roots.

### 6.9.1 Config Interface

At a high level the HTTPServer config interface needs to support the ability to list the contents of the current virtual roots as well as the ability to modify the current virtual roots. This will translate to a number of supported commands as follows:

http: //ipAddress/portal/protocols/https?Command=ListVirtualRoots

http://ipAddress/portal/protocols/https?Command=AddVirtualRoot&Name=virtualRoot&Local Root

http://ipAddress/portal/protocols/https?Command=RemoveVirtualRoot&Name=virtualRoot&LocalRoot=localRoot

The structure of the return message is the same for all commands, as all commands will return the current contents of the virtual roots table after the command is executed. The structure is as follows:

```
<?xml version="1.0"  encoding="UTF-8"    standalone="yes" ?>

<VIRTUAL_ROOTS version=" 1.0">
```

```
<VIRTUAL_ROOT        name=" virtualRootName"
                     localRoot=" localRootName"/>
<VIRTUAL_ROOT        name=" virtualRootName"
                     localRoot=" localRootName"/>
<VIRTUAL_ROOT        name=" virtualRootName"
                     localRoot=" localRootName"/>
</ VIRTUAL_ROOTS>
```

Since the virtual roots are persistent, the AddVirtualRoot and RemoveVirtualRoot commands will need to modify the contents of the initialization file.

### 6.9.2 Initialization File

The HTTPServer.protocol initialization file, will contain the same XML format that is returned from the config edge and will look like the following:

```
<?xml version='1.0'  encoding='UTF-8'  standalone='yes' ?>
<PROTOCOL name='HTTPServer'>
    <INITFILE module='HTTPServer'>
        <VIRTUAL_ROOTS version=" 1.0">
        <VIRTUAL_ROOT        name=" virtualRootName"
                             localRoot=" localRootName"/>
        <VIRTUAL_ROOT        name=" virtualRootName"
                             localRoot=" localRootName"/>
```

<VIRTUAL_ROOT name=" virtualRootName"

localRoot=" localRootName"/>

</ VIRTUAL_ROOTS>

</ INITFILE>

</ PROTOCOL>

## *6.10 PortalSock*

PortalSock supports a Config Interface on which it receives synchronous commands from AudioSwitch to add or remove the given host to or from a multicast group.

### 6.10.1 Config Interface

The PortalSock config interface needs to support the ability to add the host it is running on to a given multicast group, the ability to remove the host from the multicast group and to report which multicast groups the host is currently a part of. This will translate to the following supported commands:

http://ipAddress/portal/protocols/PortalSock?command=JoinMulticastGroup&IpAddress=224.0.0.XX

http://ipAddress/portal/protocols/PortalSock?command=LeaveMulticastGroup&IpAddress=224.0.0.XX

http://ipAddress/portal/protocols/PortalSock?command=ListMulticastGroups

All three commands should return a current list of the multicast groups that the host is currently part of after the command completes, as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<MULTICAST_GROUPS version="1.0" >
    <IP_ADDRESS value=" 224.0.0.xx" />
    <IP_ADDRESS value=" 224.0.0.xx" />
</MULTICAST_GROUPS>
```

## 6.11 Playlist Parsers

All Playlist Parsers (i.e. M3Uparse, ASXParse, PLSParse, etc.) will take in the contents of the content specific playlist file, via an HTTP request on their data edge, and output an XML message with the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<FILELIST version=" 1.0" >
    <FILE fullName=" localName" />
    <FILE fullName=" localName" />
    <FILE fullName=" localName" />
</FILELIST>
```

Note that the Playlist Parsers are only responsible for filling in the local file names.

## 6.12 FileGlobalizer

The FileGlobalizer will take in XML data using the FILELIST element, as output by the Playlist Parsers and is responsible for adding the url attribute, by using the VIRTUAL_ROOTS available via HTTPServer's config edge. All urls added by FileGlobalizer should be encoded for correct transport. The output message of the FileGlobalizer data edge is as follows:

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

FILELIST version=" 1.0" >

<FILE *url*=" linkName" *fullName*=" localName" />

<FILE *url*=" linkName" *fullName*=" localName" />

<FILE *url*=" linkName" *fullName*=" localName" />

</ FILELIST>

## 6.13 Heartbeat

The Heartbeat protocol will run on all hosts and broadcast a message to discovery on startup announcing the availability of the host. The feature that the host is announcing, will be determined by the channel that the Heartbeat broadcasts over. The contents of the message will indicate whether the feature is available on the given host. Which features are available will be specified in the Heartbeat initialization file. The format of the message sent to discovery on startup is as follows:

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

```
<HEARTBEAT_MESSAGE version="1.0" >

    <FEATURE_AVAILABLE/>

    <FEATURE_NOT_AVAILABLE/>

</ HEARTBEAT_MESSAGE>
```

The message must contain either the FEATURE_AVAILABLE tag or the FEATURE_NOT_AVAILABLE tag, but not both.

The format of the Heartbeat initialization file is as follows:

```
<?xml  version="1.0"  encoding="UTF-8"  standalone="yes" ?>

<PROTOCOL name='Heartbeat'>

    <INITFILE module='Heartbeat'>

        <NAMEVALUEPAIR name='Channel'      value='#' />

    </ INITFILE>

</PROTOCOL>
```

The supported channels are as follows:

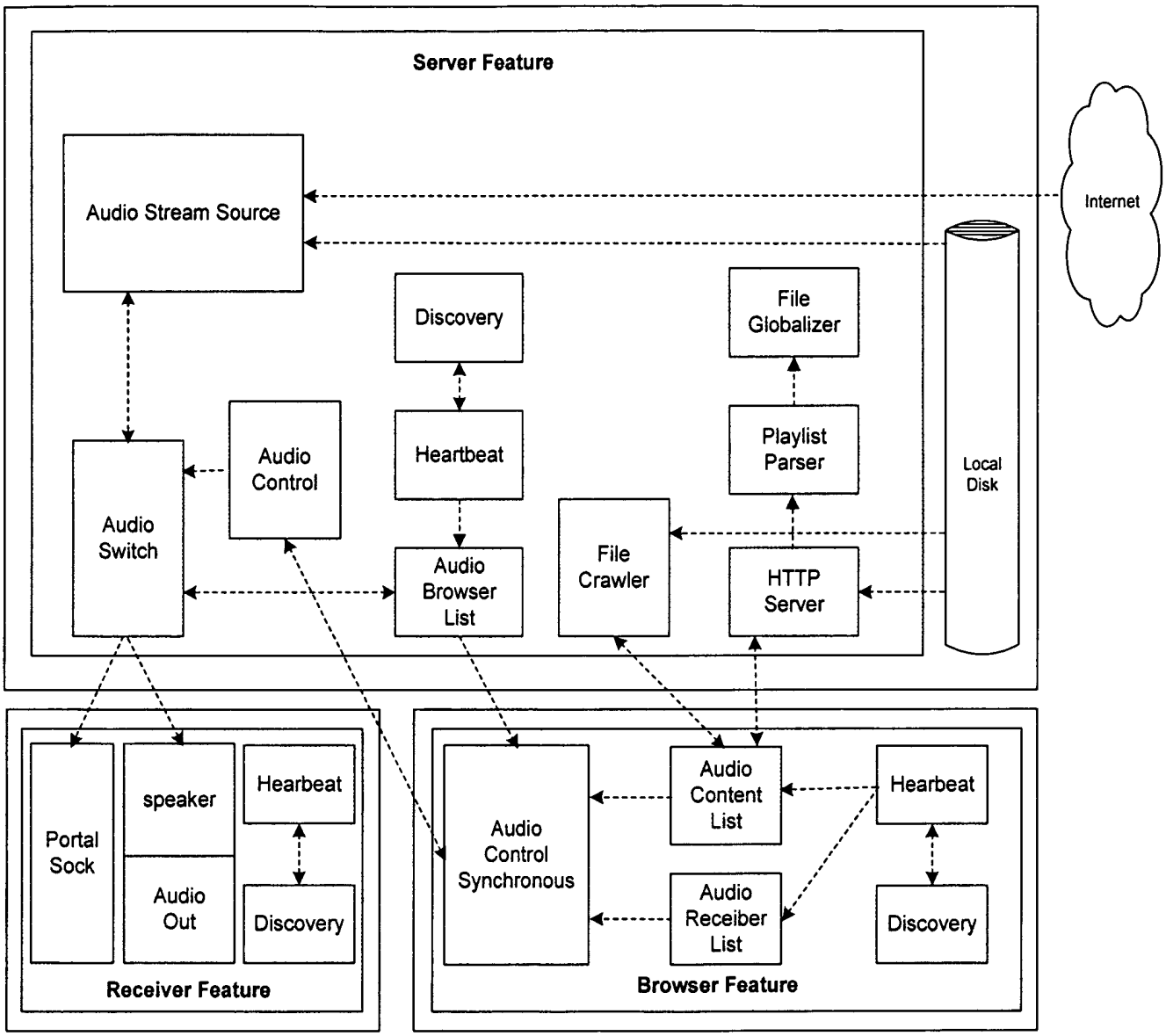
| | | |
|---|---|---|
| Channel | 3 | Local Server |
| Channel | 4 | Internet Server |
| Channel | 5 | Receiver |
| Channel | 6 | Browser |

The Heartbeat initialization file can contain multiple channels if the host is supporting multiple features, such as Local Server and Internet Server.

A component architecture overview is shown again in Figure 2.

[[

## Component Architecture Overview




]]

| Name | Parameters | Action |
|---|---|---|
| AudioContentList Config Interface | | |
| ListContent | N/A | Return message with all of the content currently available across all discovered host PCs. |
| ListHosts | N/A | Return message will all hosts server hosts currently available. |
| ListVirtualRoots | Host | Return message with all virtual roots accessible on the given host. |
| Rescan | N/A | Rescan all hosts for new content. |
| Rescan | Host | Rescan host for new content. |
| Rescan | Host<br>VirtualRoot | Rescan host specific virtual root for new content. |
| AudioContentList Event Interface | | |
| AddHosts | Host [, Host, Host, ...] | Send SERVER_EVENT and CONTENT_EVENT to AudioControlSynchronous for new servers. |
| RemoveHosts | Host [, Host, Host, ...] | Send SERVER_EVENT and CONTENT_EVENT to AudioControlSynchronous for removed servers. |
| AudioReceiverList Config Interface | | |
| ListReceivers | N/A | Return message with all of the discovered receivers. |
| AudioReceiverList Event Interface | | |
| AddHosts | Host [, Host, Host, ...] | Send RECEIVER_EVENT to AudioControlSynchronous for new servers. |
| RemoveHosts | Host [, Host, Host, ...] | Send RECEIVER_EVENT to AudioControlSynchronous for removed servers. |
| FileCrawler Config Interface | | |
| ListFiles | Root<br>Extension | Return message with all of the files under the virtual root directory that match the given extension. |

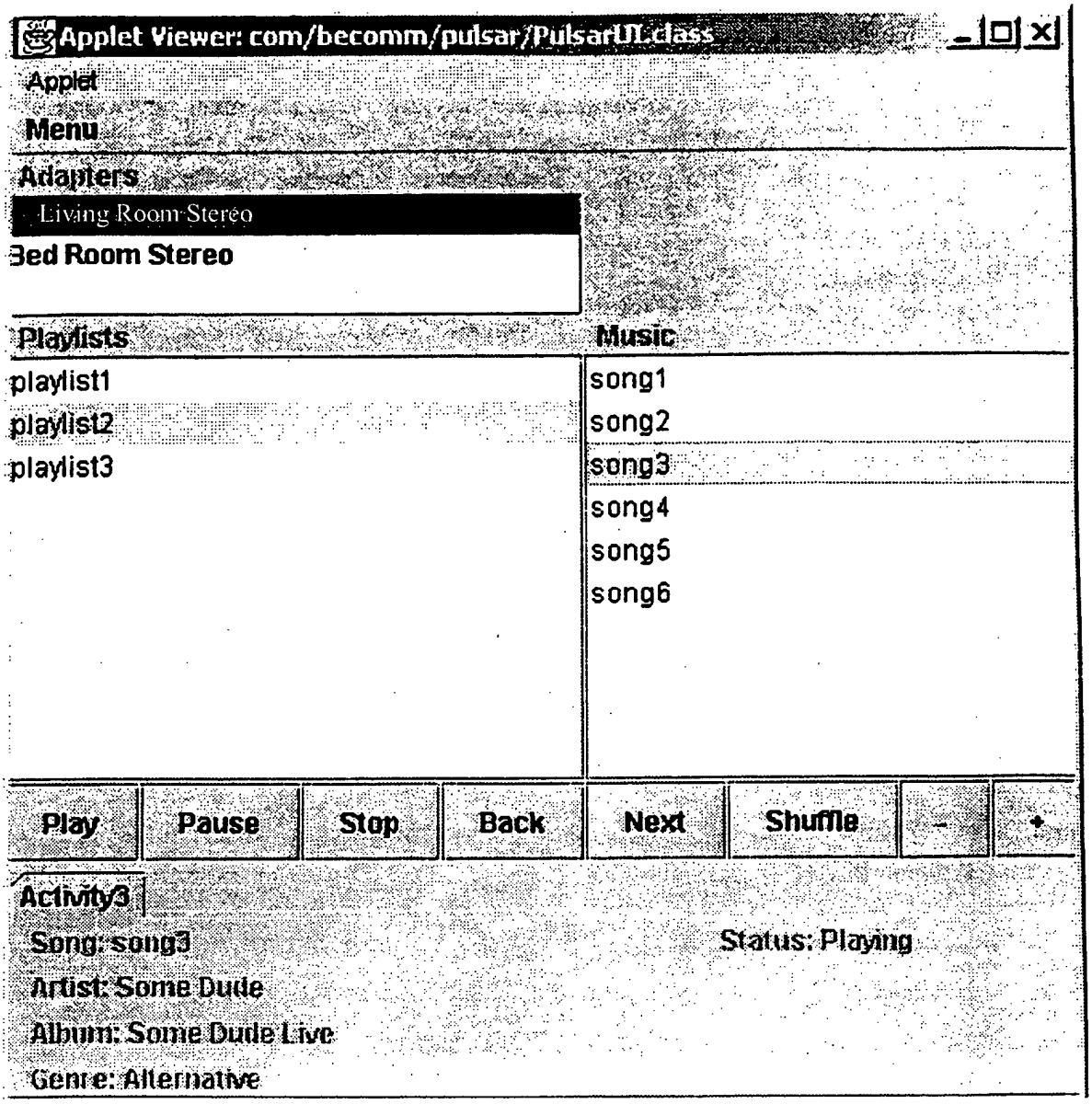
~~These~~ The images as shown in Figures 3-5 represent screen shots from an early version of BeComm's Audio Browser. The Audio Browser allows someone to control multiple audio devices and access all of the audio data on those devices.

~~This~~ The first image of Figure 3 shows the layout of the Audio Browser. A song titled "song3" is currently playing on the "Living Room Stereo". The Audio Browser should follow along a playlist by highlighting the currently playing song.

Bed Room Stereo is currently not in use and can be added simply by clicking on it. This will cause music to start playing in the Bed Room that was already playing in the Living Room. Removing an Audio player is also simply done by mouse click.

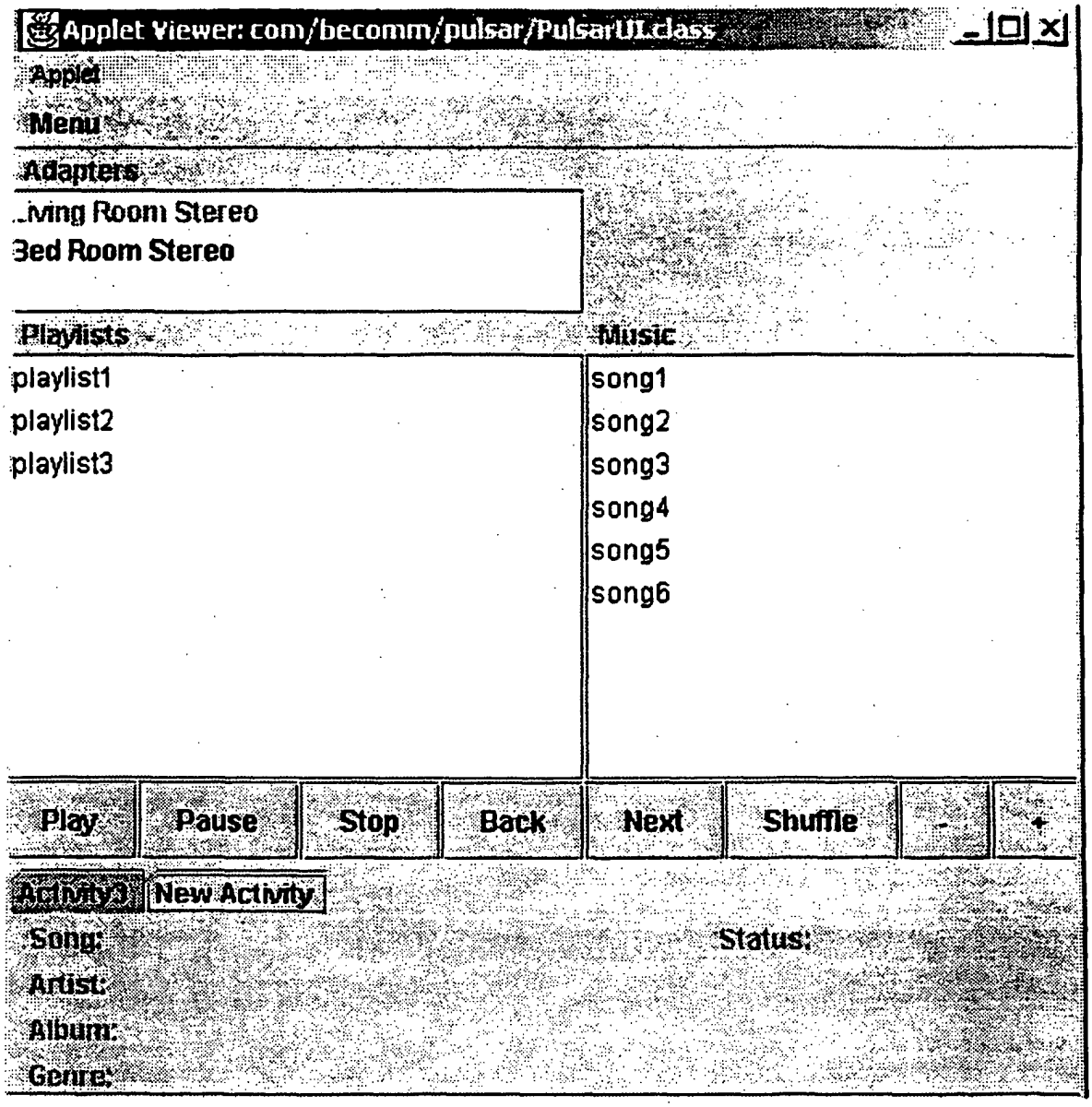
A new song or playlist could be switched at any point by double clicking on one.

[[


Applet Viewer: com/becomm/pulsar/PulsarUI.class
Applet
Menu
Adapters
Living Room Stereo
Bed Room Stereo
Playlists
Music
playlist1
playlist2
playlist3
song1
song2
song3
song4
song5
song6
Play
Pause
Stop
Back
Next
Shuffle
Activity3
Song: song3
Status: Playing
Artist: Some Dude
Album: Some Dude Live
Genre: Alternative

]]

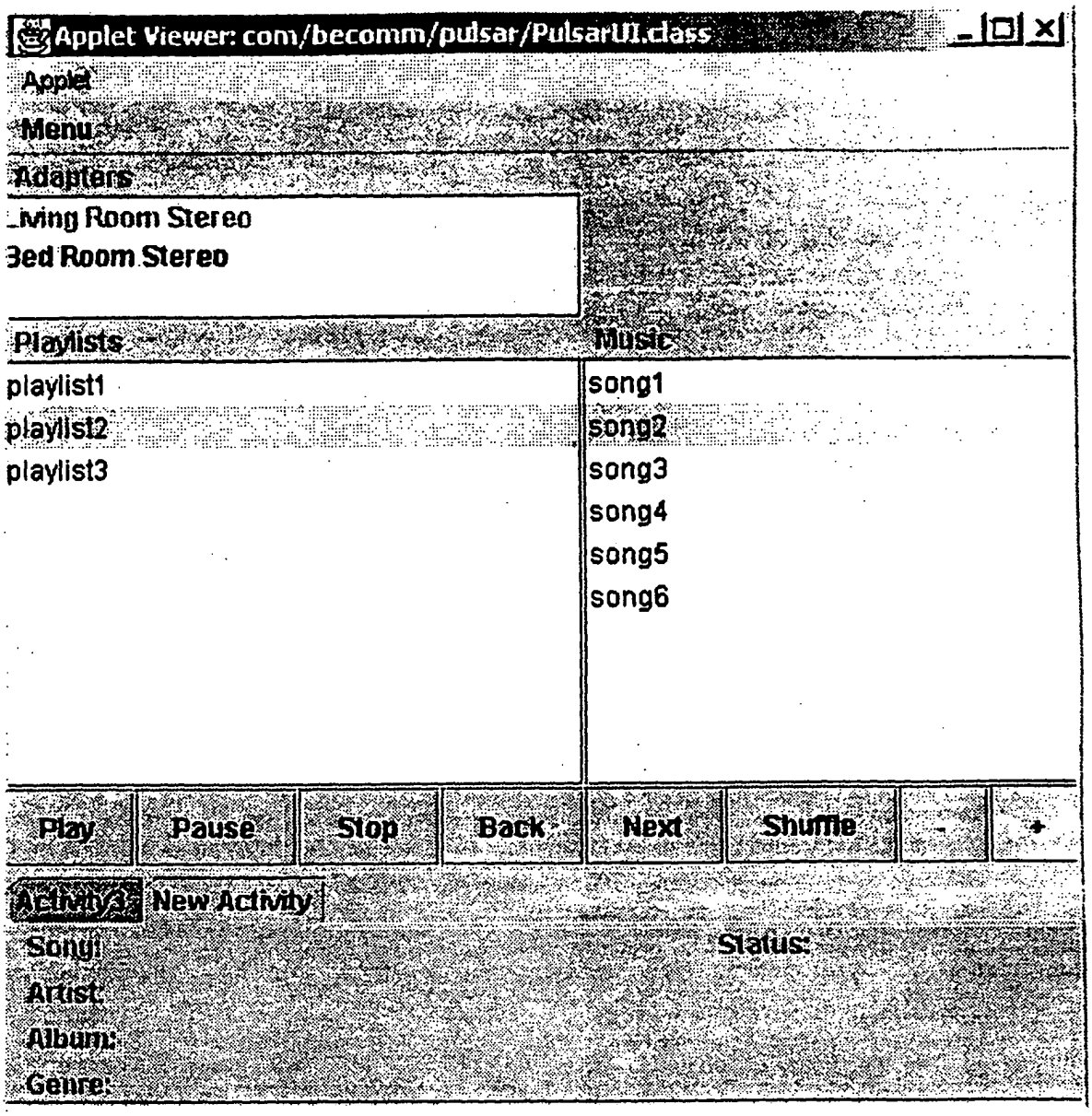
Different audio can play on different adapters at the same time. This is done by clicking on the + sign on the far right of the button panel. Now a new audio session is created. To start playing music the user picks a song, selects any set of the available adapters (currently light gray represents adapters in use), and hits the play button. [[




]]

~~This~~ The screen as shown in Figure 5 shows the Audio Browsers state before the play button is pressed. When it is song2 will start playing in the Bed Room.

[[




]]

## *AudioStream Source Events*

This document is presented to clearly specify the events generated by AudioStreamSource (ASS). Where the Component Communication document is flexible, this document is slightly more rigid specification to be used for implementation of the ASS events, specifying likely order (comparison of Real and WMA needs to be done) and exact message contents. Note that buffering can take place at any time, except while stopped with no outstanding commands.

For a standard **open-play** scenario of an mp3 format file, following are the events generated by the AudioStreamSource protocol, in order:

[Open command issued ]

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version=" 1.0">

<LENGTH value=" 0" units="milliseconds" />

</ PLAYBACK_EVENT>

**Not implemented yet:**

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<TITLE name="Title" />

<ARTIST name="Artist" />

<ALBUM name="Album" />

<GENRE name=" Genre" />

</ PLAYBACK_EVENT>

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = Stopped" />

</ PLAYBACK_EVENT>

[Play command Issued ]

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = "Playing" />

</PLAYBACK_EVENT>

**Net congestion:**

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = "Buffering" />

<PROGRESS value=" 0-100" units="percentage" />

</ PLAYBACK_EVENT>

[song plays to completion]

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = "Stopped" />

</ PLAYBACK_EVENT>

For a standard **seek** scenario of an mp3 format file, following are the events generated by the AudioStreamSource protocol, in order:

[Seek command issued ]

**Seek:**

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = "Buffering" />

<PROGRESS value=" 0-100" units="percentage" />

</PLAYBACK_EVENT>

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = "Playing" />

</ PLAYBACK_EVENT>

For a standard **pause of a live stream** scenario of an mp3 format file, following are the events generated by the AudioStreamSource protocol, in order:

[Pause command issued ]

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = "Paused" />

</PLAYBACK_EVENT>

[Resume command issued ]

**Resume live stream:**

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = "Buffering" />

<PROGRESS value=" 0-100" units="percentage" />

</ PLAYBACK_EVENT>

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = "Playing" />

</ PLAYBACK_EVENT>

For a standard **stop-play** scenario of an mp3 format file, following are the events generated by the AudioStreamSource protocol, in order:

[Stop command issued]

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = "Stopped" />

</ PLAYBACK_EVENT>

[Play command issued]

**Net congestion:**

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

<STATE value = "Buffering" />

<PROGRESS value=" 0-100" units="percentage" />

</ PLAYBACK_EVENT>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<PLAYBACK_EVENT version="1.0" >

        <STATE value = "Playing" />

</ PLAYBACK_EVENT>
```

## CLAIM

WHAT IS CLAIMED IS:

1. An audio server system for streaming audio content, comprising:

an audio switch that receives commands from an audio control component and sends commands to an audio stream source, that receives events from an audio stream source and sends events to an audio browse list component, and that controls the switching of audio from the audio stream source to one or more receivers; and

an audio control component that receives commands from an audio control synchronous component and sends commands to the audio switch.

ABSTRACT

An audio server system for streaming audio content has an audio switch that receives commands from an audio control component and sends commands to an audio stream source, that receives events from an audio stream source and sends events to an audio browse list component, and that controls the switching of audio from the audio stream source to one or more receivers; and an audio control component that receives commands from an audio control synchronous component and sends commands to the audio switch.